CSC 311 – Winter 2022-2023
Analysis and Design of Algorithms
8. Graph Algorithms

Prof. Mohamed Menai
Department of Computer Science
King Saud University

Outline

- Representing graphs
- Graph searching
- Breadth-First Search
- Depth-First Search

2

Graphs

- A graph $G = (V, E)$
  - V = set of vertices
  - E = set of edges = subset of $V \times V$
  - Thus $|E| = O(|V|^2)$

Graph Variations

- Variations:
  - A *connected graph* has a path from every vertex to every other
  - In an *undirected graph*
    - edge (u,v) = edge (v,u)
    - No self-loops
  - In a *directed* graph:
    - edge (u,v) goes from vertex u to vertex v, notated $u \rightarrow v$

Graph Variations

- More variations:
  - A *weighted graph* associates weights with either the edges or the vertices
    - e.g., a road map: edges might be weighted w/ distance
  - A *multigraph* allows multiple edges between the same vertices
    - e.g., the call graph in a program (a function can get called from multiple points in another function)

Graphs

- We will typically express running times in terms of $|E|$ and $|V|$ (often dropping the |'s)
  - If $|E| \approx |V|^2$ the graph is *dense*
  - If $|E| \approx |V|$ the graph is *sparse*
- If you know you are dealing with dense or sparse graphs, different data structures may make sense

6

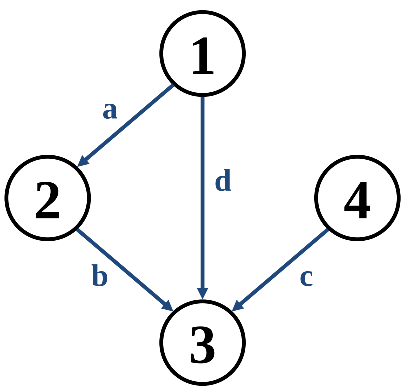
Representing Graphs

- Assume $V = \{1, 2, \ldots, n\}$
- An *adjacency matrix* represents the graph as a n x n matrix A:
  - $A[i, j]$ = 1 if edge $(i, j) \in E$ (or weight of edge)
    = 0 if edge $(i, j) \notin E$

7

Graphs: Adjacency Matrix

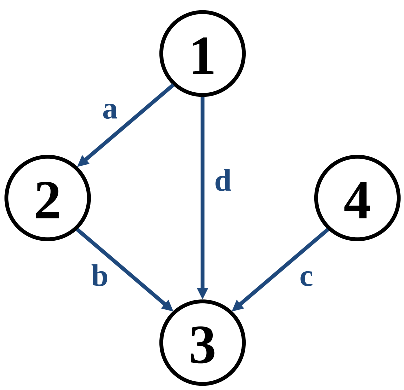
- Example:



A	1	2	3	4
1				
2				
3			??	
4				

Graphs: Adjacency Matrix

- Example:



A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

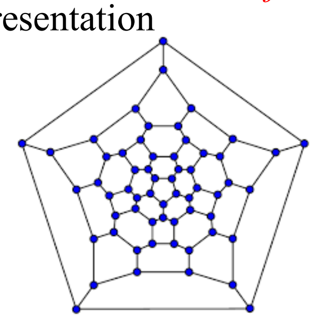
9

Graphs: Adjacency Matrix

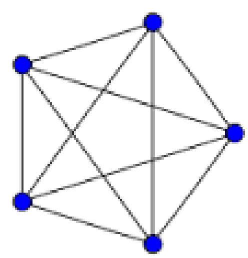
- *How much storage does the adjacency matrix require?*
- A: $O(V^2)$
- *What is the minimum amount of storage needed by an adjacency matrix representation of an undirected graph with 4 vertices?*
- A: 6 bits
  - Undirected graph $\rightarrow$ matrix is symmetric
  - No self-loops $\rightarrow$ don't need diagonal

Graphs: Adjacency Matrix

- The adjacency matrix is a dense representation
  - Usually too much storage for large graphs
  - But can be very efficient for small graphs
- Most large interesting graphs are sparse
  - e.g., planar graphs, in which no edges cross, have $|E| = O(|V|)$ by Euler's formula
  - For this reason the *adjacency list* is often a more appropriate representation



Planar graph

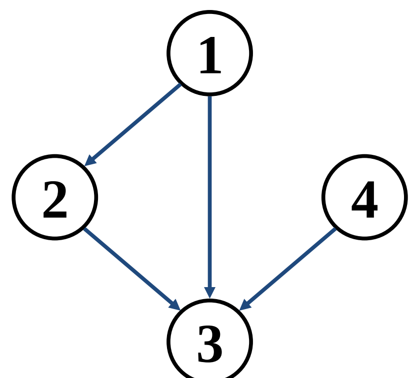


Non planar graph

11

Graphs: Adjacency List

- Adjacency list: for each vertex $v \in V$, store a list of vertices adjacent to v
- Example:
  - Adj[1] = {2,3}
  - Adj[2] = {3}
  - Adj[3] = {}
  - Adj[4] = {3}
- Variation: can also keep a list of edges coming *into* vertex



12

Graphs: Adjacency List

- How much storage is required?
  - The *degree* of a vertex v = # incident edges
    - Directed graphs have in-degree, out-degree
  - For directed graphs, # of items in adjacency lists is
    Σ out-degree(v) = $|E|$
    takes $\Theta(V + E)$ storage
  - For undirected graphs, # items in adjacency lists is
    Σ degree(v) = 2 $|E|$ (*handshaking lemma*)
    also $\Theta(V + E)$ storage
  - (In a party of people some of whom shake hands, an even number of people must have shaken an odd number of other people's hands).
- So: Adjacency lists take $O(V+E)$ storage

13

Graph Searching

- Given: a graph $G = (V, E)$, directed or undirected
- Goal: methodically explore every vertex and every edge
- Ultimately: build a tree on the graph
  - Pick a vertex as the root
  - Choose certain edges to produce a tree

Breadth-First Search

- "Explore" a graph, turning it into a tree
  - One vertex at a time
  - Expand frontier of explored vertices across the *breadth* of the frontier
- Builds a tree over the graph
  - Pick a *source vertex* to be the root
  - Find ("discover") its children, then their children, etc.

Breadth-First Search

- Will associate vertex colors to guide the algorithm
  - White vertices have not been discovered
    - All vertices start out white
  - Grey vertices are discovered but not fully explored
    - They may be adjacent to white vertices
  - Black vertices are discovered and fully explored
    - They are adjacent only to black and gray vertices
- Explore vertices by scanning adjacency list of grey vertices

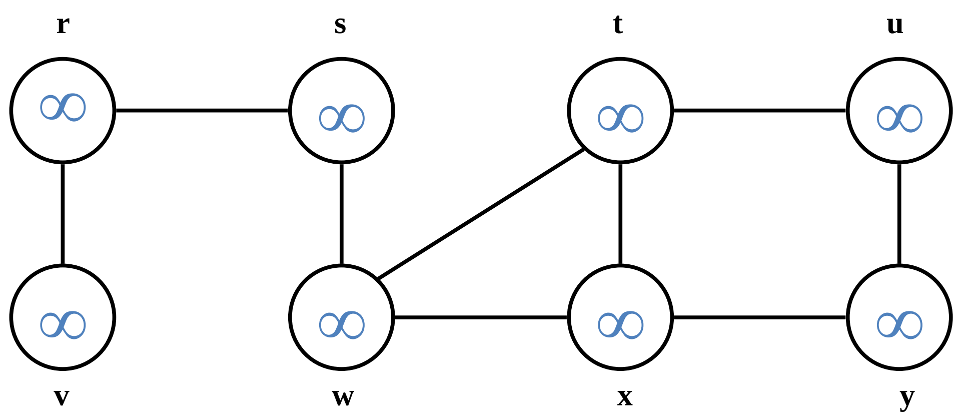
Breadth-First Search

```
BFS(G, s) {
   initialize vertices;
   Q = {s};                    // Q is a FIFO queue; initialize to s
   while (Q not empty) {
      u = Dequeue(Q);
      for each v ∈ u->adj {
         if (v->color == WHITE){
            v->color = GREY;
            v->d = u->d + 1;
            v->p = u;
            Enqueue(Q, v);}
      }
      u->color = BLACK;
   }
}
```

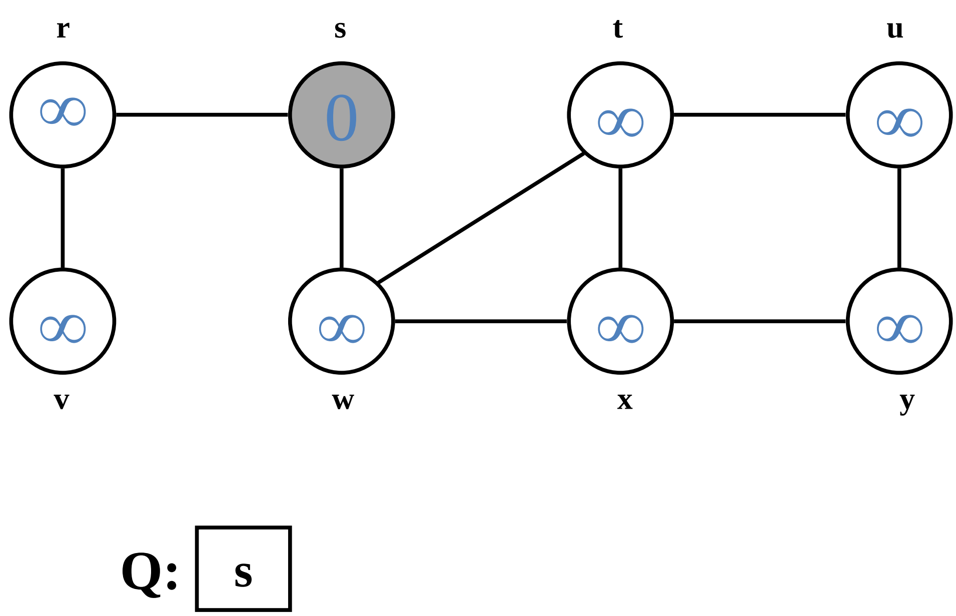
What does `v->d` represent?

What does `v->p` represent?

Breadth-First Search: Example

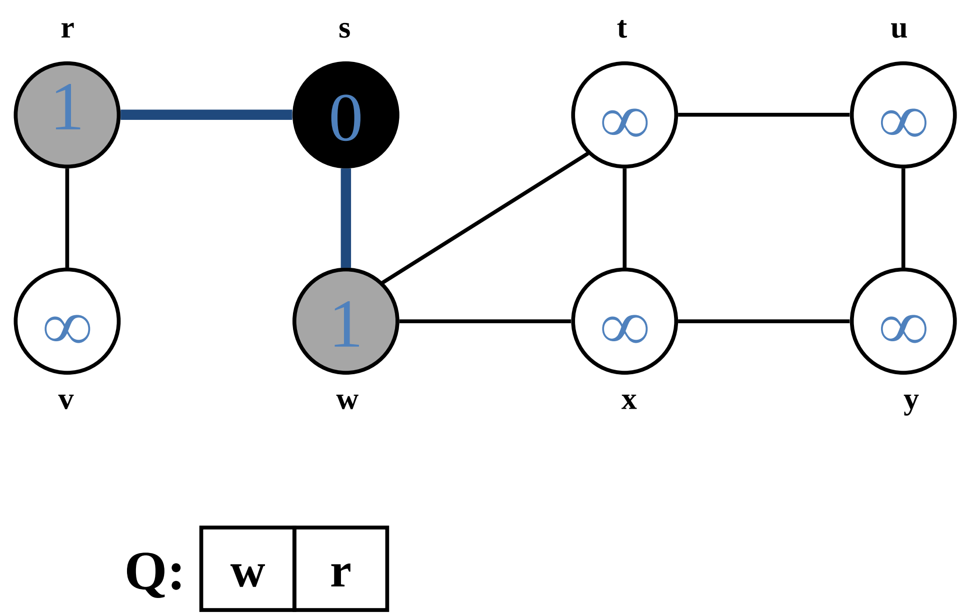


Breadth-First Search: Example



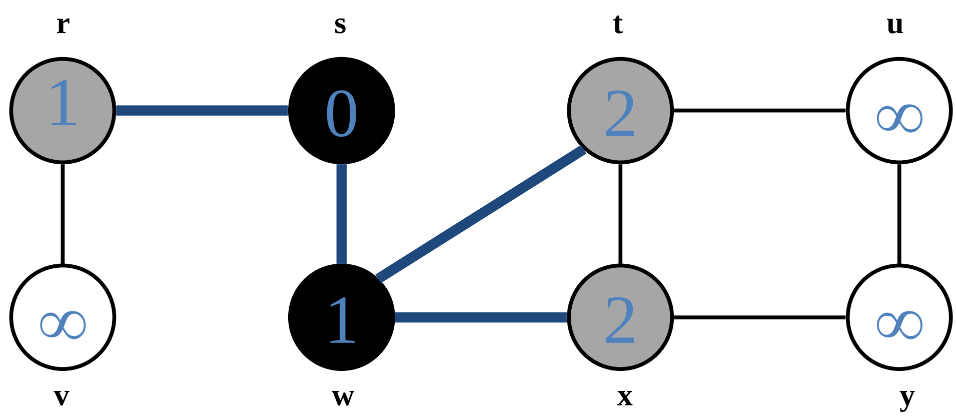
19

Breadth-First Search: Example



20

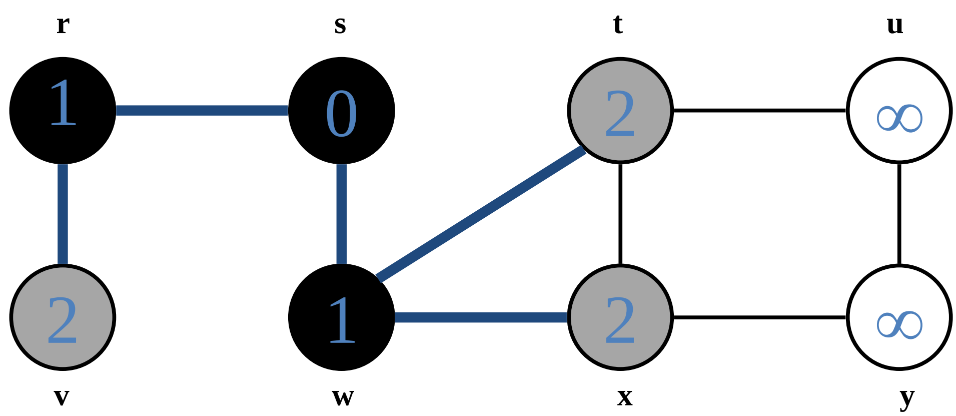
Breadth-First Search: Example



Q:	r	t	x

21

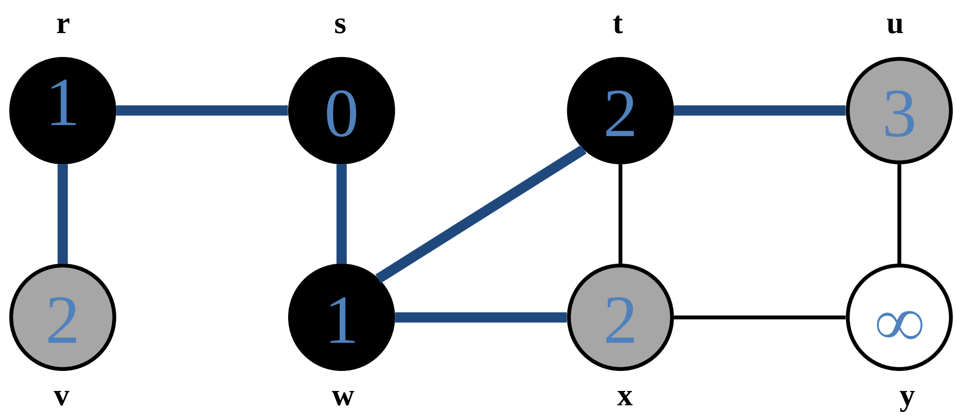
Breadth-First Search: Example



Q: | t | x | v |

22

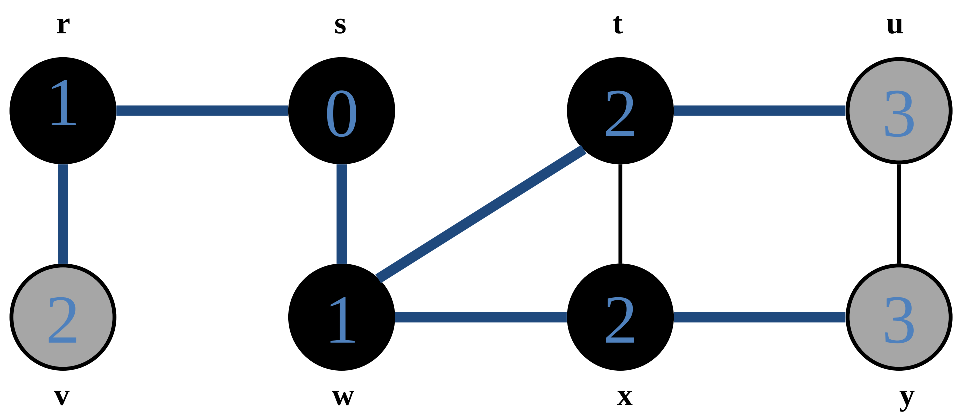
Breadth-First Search: Example



Q:

x	v	u

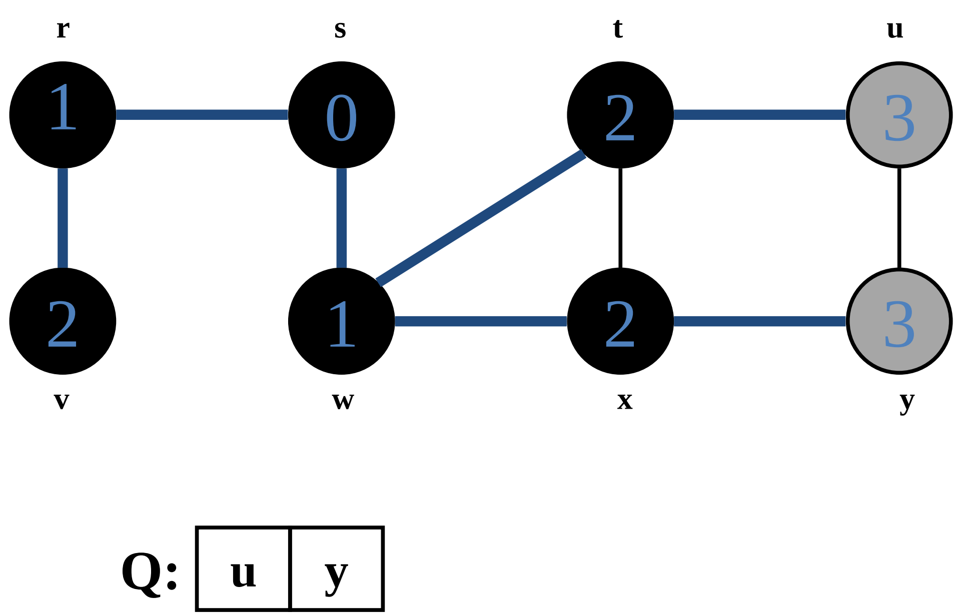
Breadth-First Search: Example



Q:

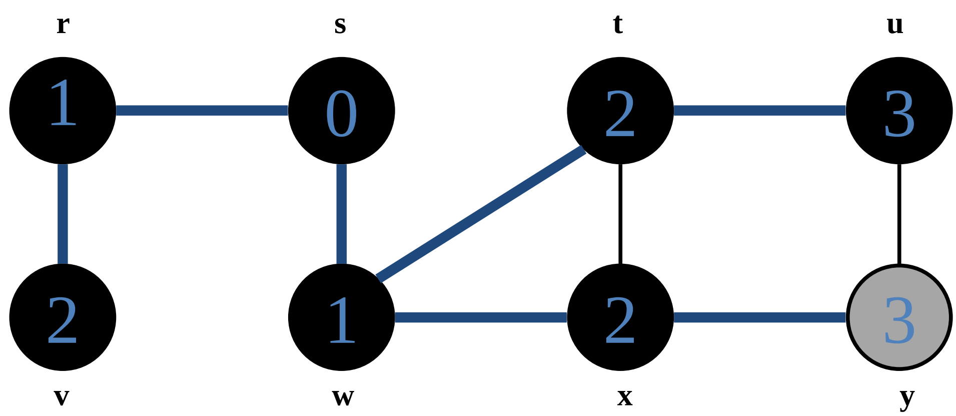
v	**u**	**y**

Breadth-First Search: Example



25

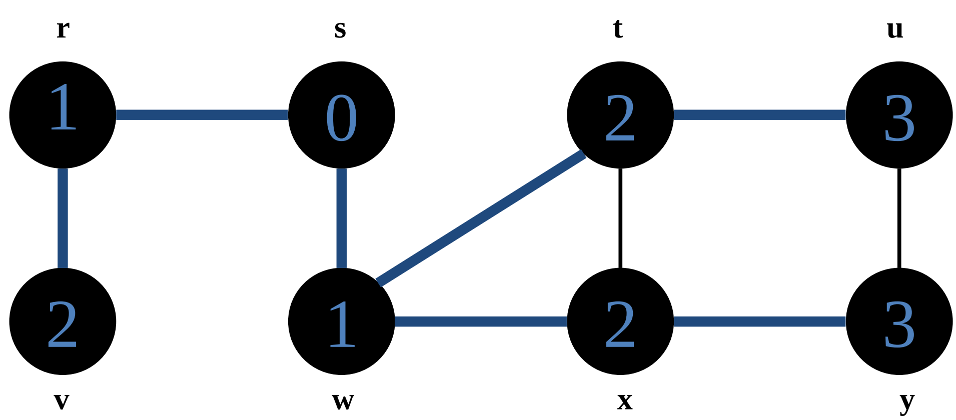
Breadth-First Search: Example



Q: y

26

Breadth-First Search: Example



Q: Ø

BFS: The Code Again

```
BFS(G, s) {
   initialize vertices;
   Q = {s};
   while (Q not empty) {
      u = Dequeue(Q);
      for each v ∈ u->adj {
         if (v->color == WHITE)
            v->color = GREY;
            v->d = u->d + 1;
            v->p = u;
            Enqueue(Q, v);
      }
      u->color = BLACK;
   }
}
```

Touch every vertex: $O(V)$

u = every vertex, but only once

v = every vertex that appears in some other vertex's adjacency list

What will be the running time?

Total running time: $O(V+E)$

BFS: The Code Again

```
BFS(G, s) {
   initialize vertices;
   Q = {s};
   while (Q not empty) {
      u = RemoveTop(Q);
      for each v ∈ u->adj {
         if (v->color == WHITE)
            v->color = GREY;
            v->d = u->d + 1;
            v->p = u;
            Enqueue(Q, v);
      }
      u->color = BLACK;
   }
}
```

What will be the storage cost in addition to storing the tree?

Total space used:
$O(\max(\text{degree}(v))) = O(E)$

Breadth-First Search: Properties

- BFS calculates the *shortest-path distance* to the source node
  - Shortest-path distance $\delta(s,v)$ = minimum number of edges from s to v, or ∞ if v not reachable from s
- BFS builds *breadth-first tree*, in which paths to root represent shortest paths in G
  - We can use BFS to calculate shortest path from one vertex to another in $O(V+E)$ time

Depth-First Search

- *Depth-first search* is another strategy for exploring a graph
  - Explore deeper in the graph whenever possible
  - Edges are explored out of the most recently discovered vertex v that still has unexplored edges
  - When all of v's edges have been explored, backtrack to the vertex from which v was discovered

Depth-First Search

- Vertices initially colored white
- Then colored gray when discovered
- Then black when finished

Depth-First Search: The Code

```
DFS(G)
{
  for each vertex u ∈ G->V
  {
    u->color = WHITE;
  }
  time = 0;
  for each vertex u ∈ G->V
  {
    if (u->color == WHITE)
      DFS_Visit(u);
  }
}
```

```
DFS_Visit(u)
{
  u->color = GREY;
  time = time+1;
  u->d = time;
  for each v ∈ u->Adj[]
  {
    if (v->color == WHITE)
      DFS_Visit(v);
  }
  u->color = BLACK;
  time = time+1;
  u->f = time;
}
```

33

Depth-First Search: The Code

```
DFS(G)
{
  for each vertex u ∈ G->V
  {
    u->color = WHITE;
  }
  time = 0;
  for each vertex u ∈ G->V
  {
    if (u->color == WHITE)
      DFS_Visit(u);
  }
}
```

```
DFS_Visit(u)
{
  u->color = GREY;
  time = time+1;
  u->d = time;
  for each v ∈ u->Adj[]
  {
    if (v->color == WHITE)
      DFS_Visit(v);
  }
  u->color = BLACK;
  time = time+1;
  u->f = time;
}
```

What does u->d represent?

34

Depth-First Search: The Code

```
DFS(G)
{
  for each vertex u ∈ G->V
  {
    u->color = WHITE;
  }
  time = 0;
  for each vertex u ∈ G->V
  {
    if (u->color == WHITE)
      DFS_Visit(u);
  }
}
```

```
DFS_Visit(u)
{
  u->color = GREY;
  time = time+1;
  u->d = time;
  for each v ∈ u->Adj[]
  {
    if (v->color == WHITE)
      DFS_Visit(v);
  }
  u->color = BLACK;
  time = time+1;
  u->f = time;
}
```

What does `u->f` represent?

Depth-First Search: The Code

- u->d: timestamp that records when u is first discovered (and then grayed).
- u->f: timestamp that records when the search finishes examining adjacency list of u (and blackens)

36

Depth-First Search: The Code

```
DFS(G)
{
  for each vertex u ∈ G->V
  {
    u->color = WHITE;
  }
  time = 0;
  for each vertex u ∈ G->V
  {
    if (u->color == WHITE)
      DFS_Visit(u);
  }
}
```

```
DFS_Visit(u)
{
  u->color = GREY;
  time = time+1;
  u->d = time;
  for each v ∈ u->Adj[]
  {
    if (v->color == WHITE)
      DFS_Visit(v);
  }
  u->color = BLACK;
  time = time+1;
  u->f = time;
}
```

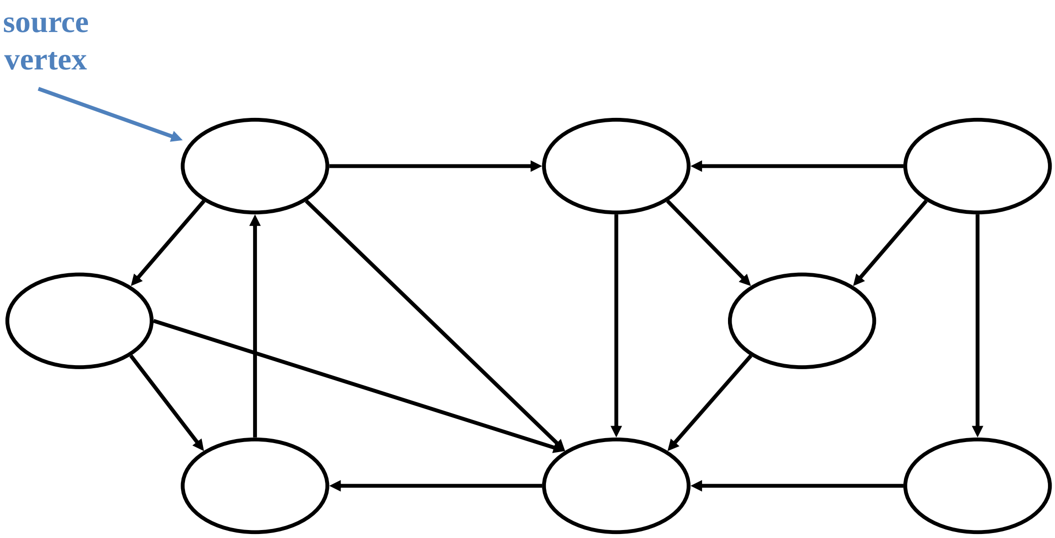
What will be the running time?

37

Depth-First Search Analysis

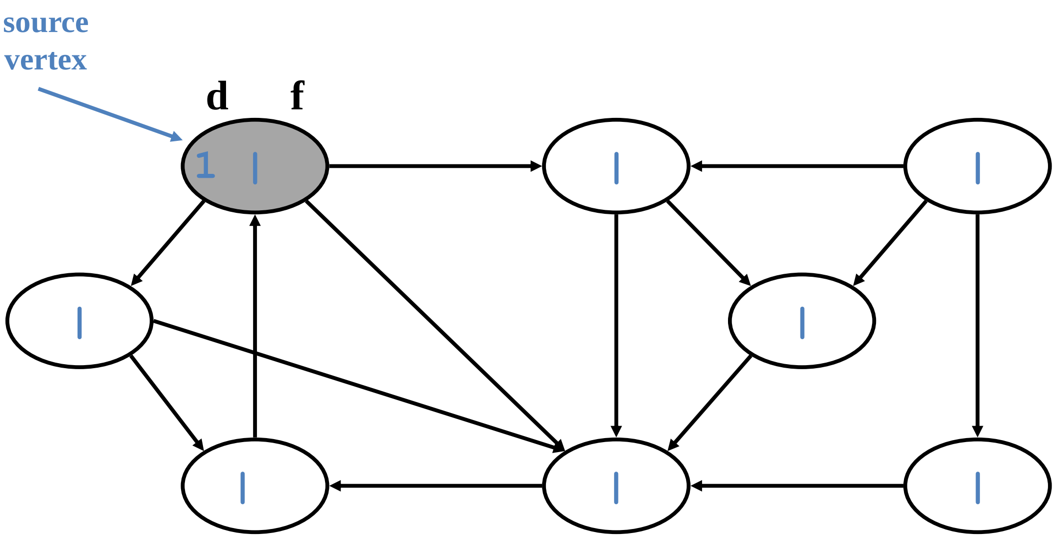
- Running time:
  - The exploration of edge to edge:
    - Each loop in DFS_Visit can be attributed to an edge in the graph
    - Runs once/edge if directed graph, twice if undirected
    - Thus loop will run in $O(E)$ time, algorithm $O(V+E)$

DFS Example

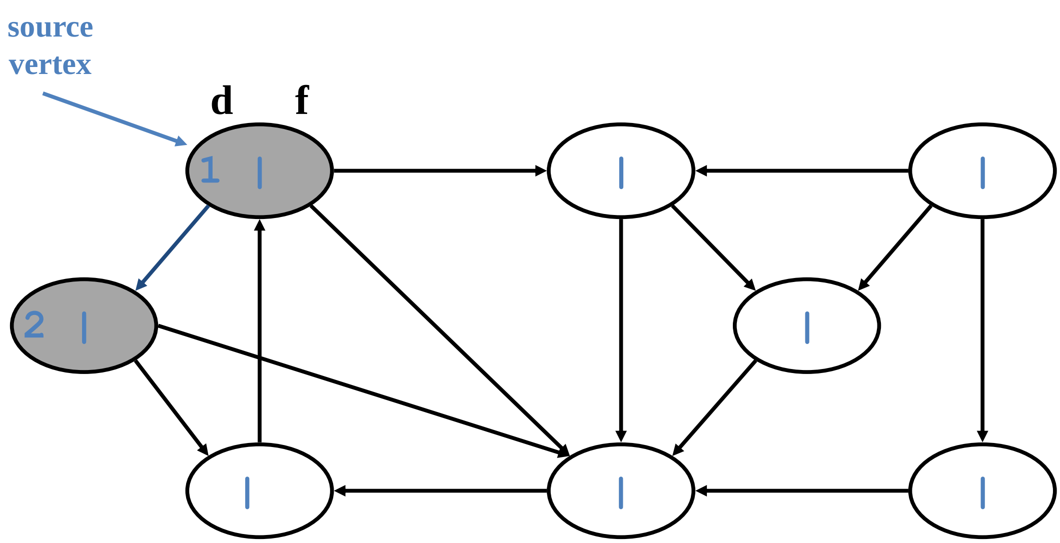


39

DFS Example

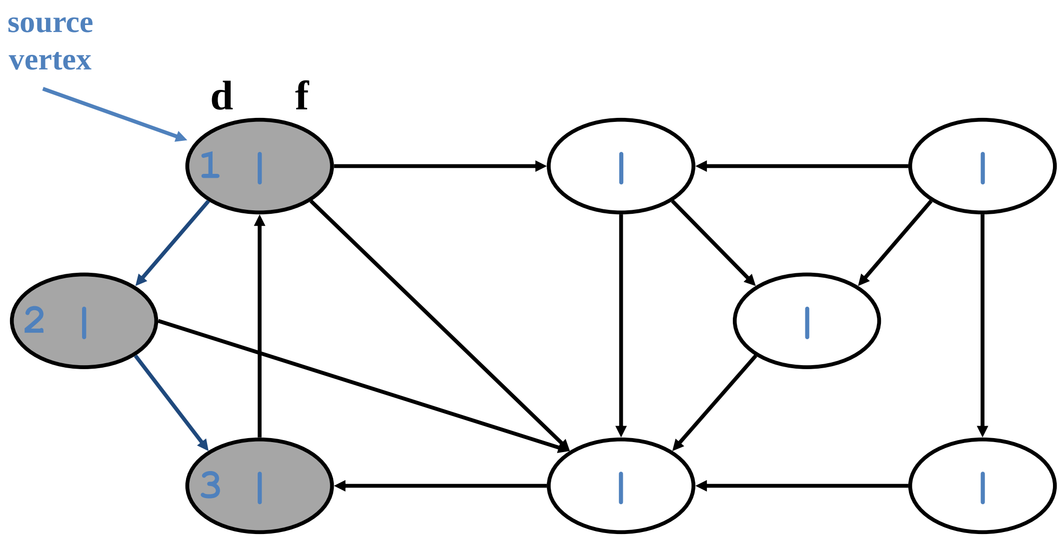


DFS Example

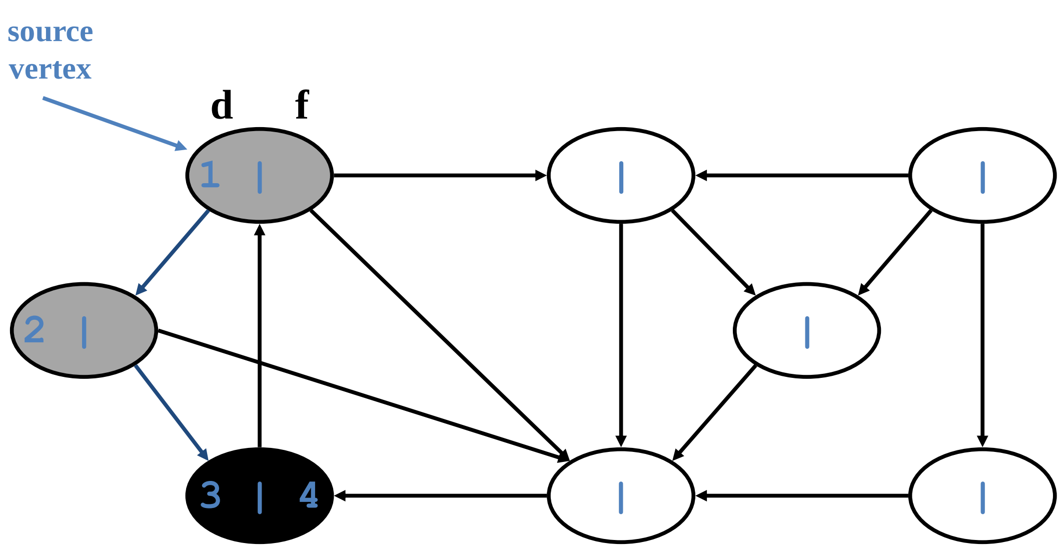


41

DFS Example

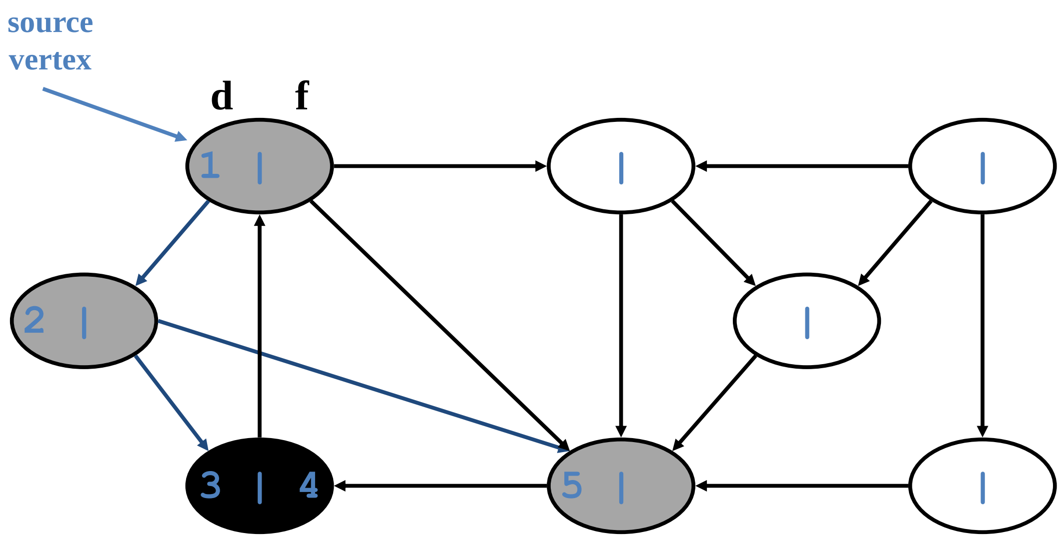


DFS Example



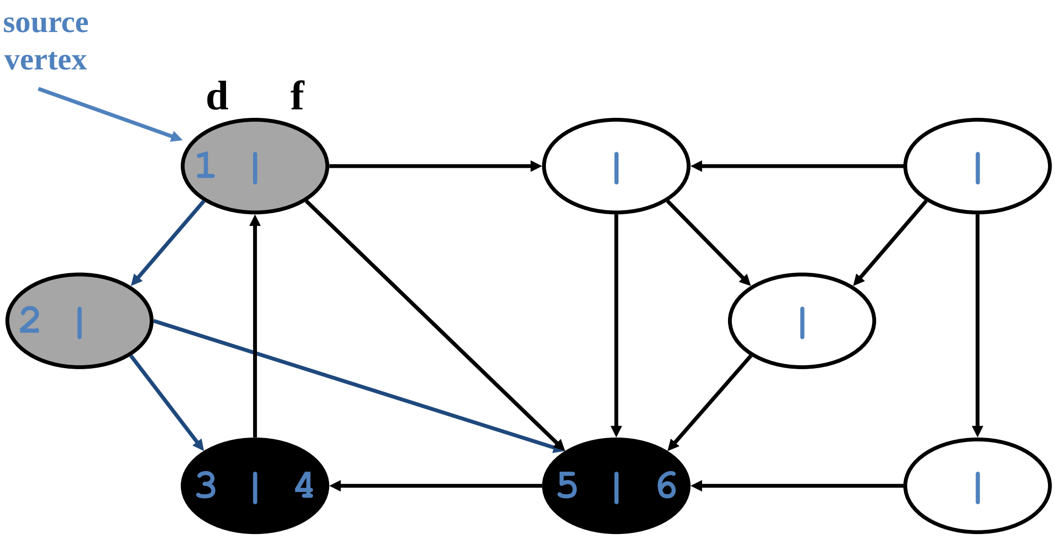
43

DFS Example



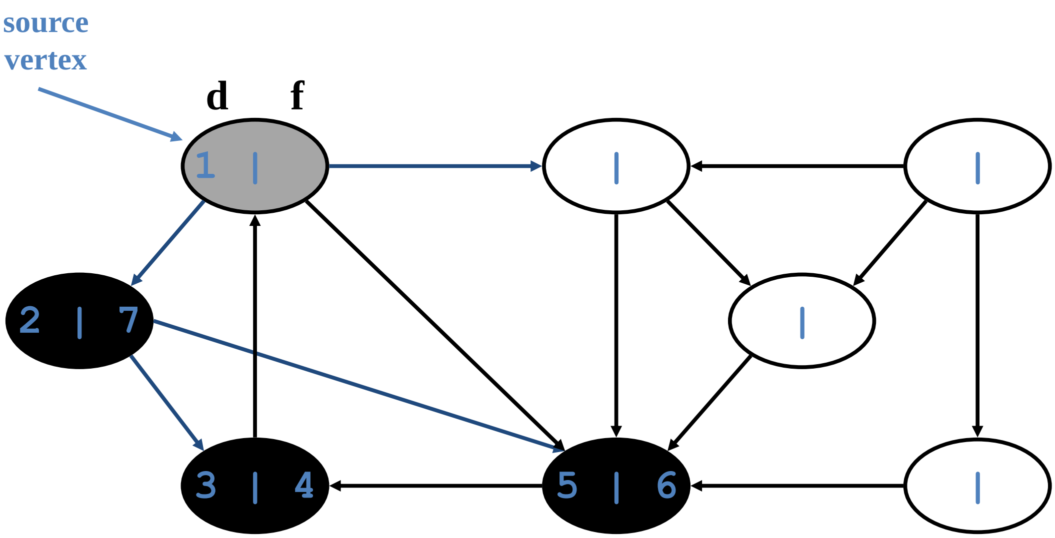
44

DFS Example



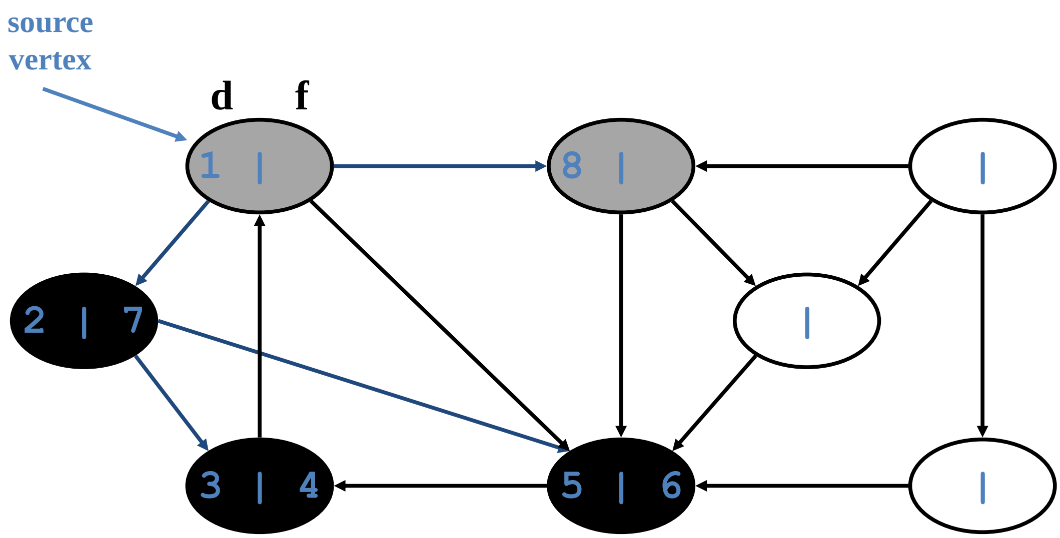
45

DFS Example

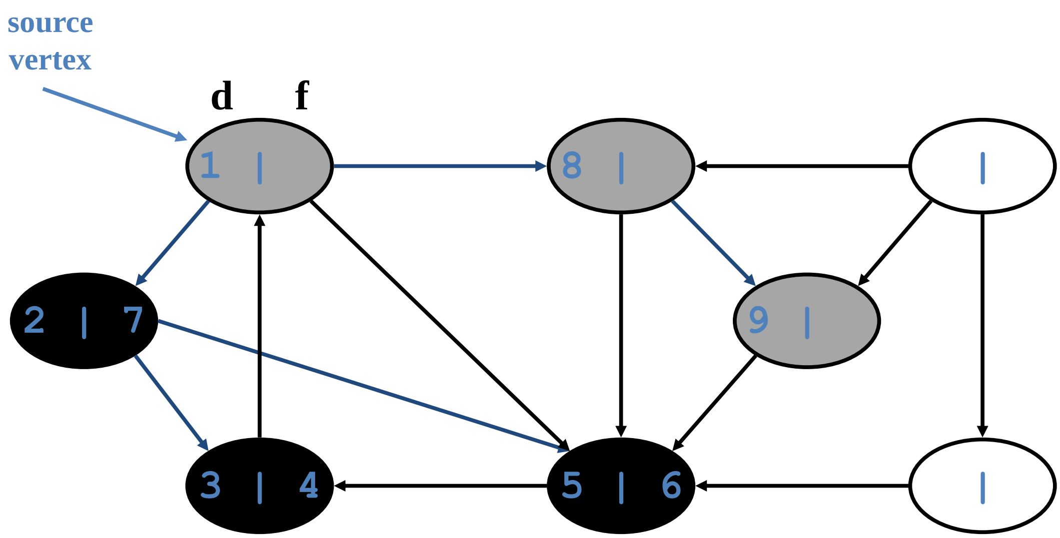


46

DFS Example

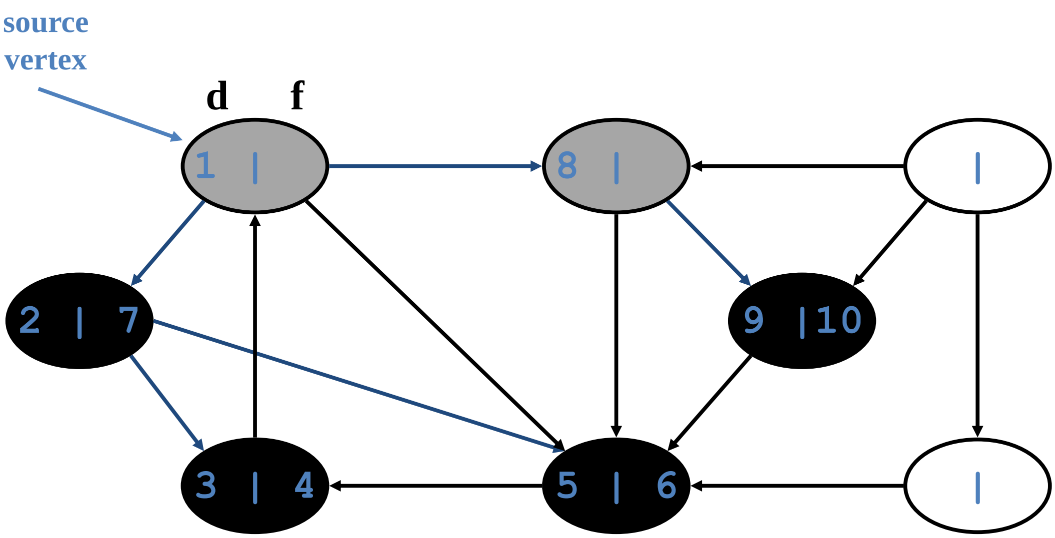


DFS Example



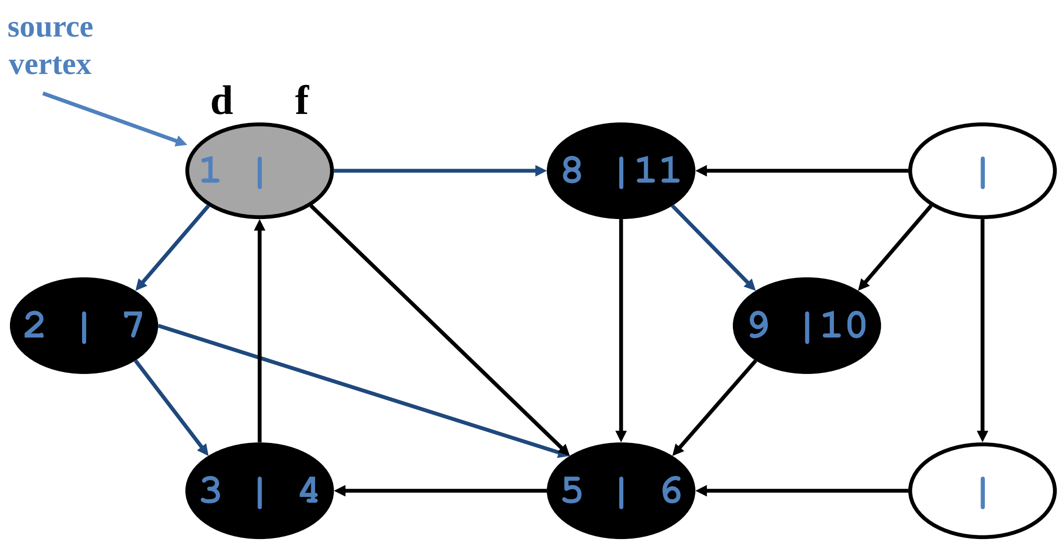
48

DFS Example



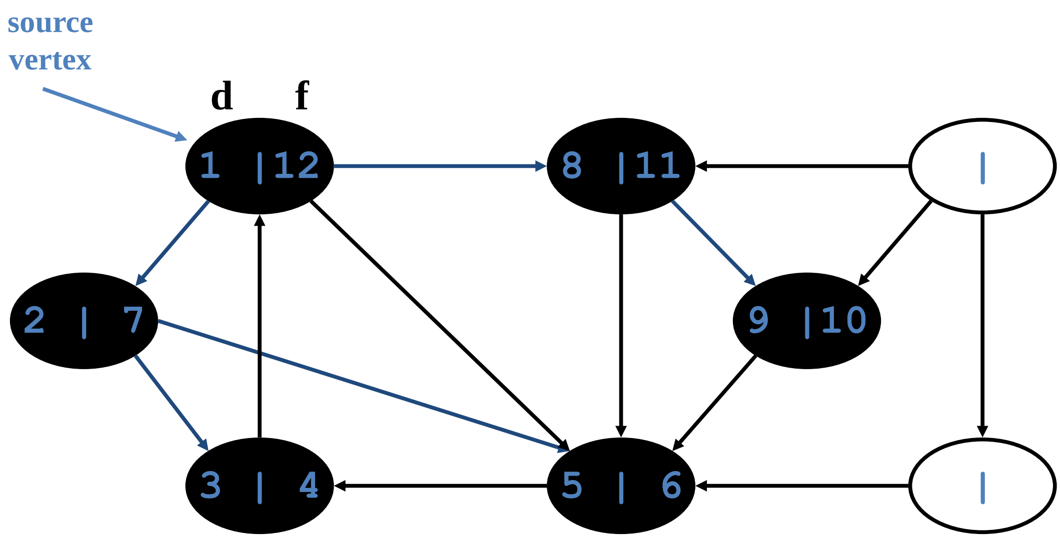
49

DFS Example



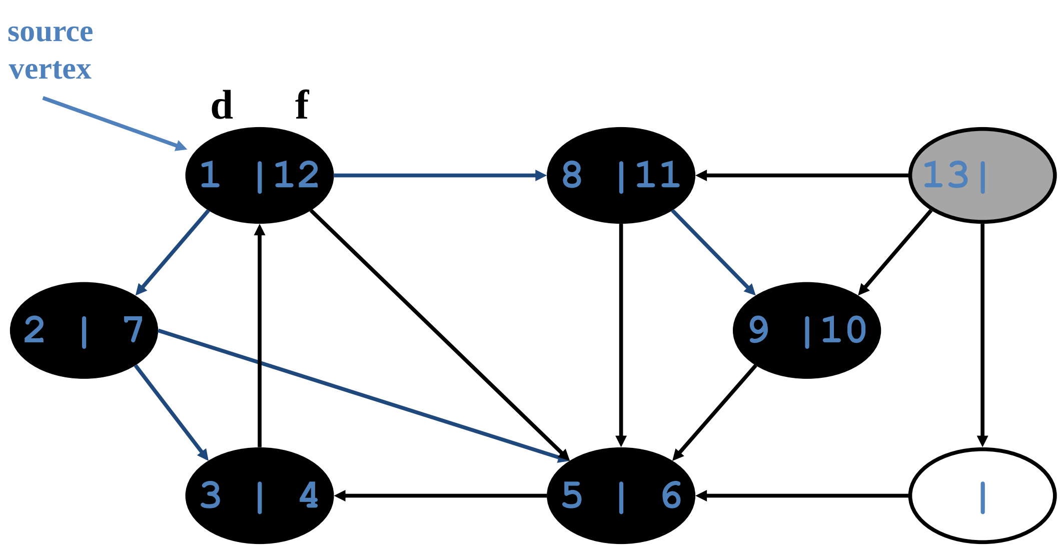
50

DFS Example



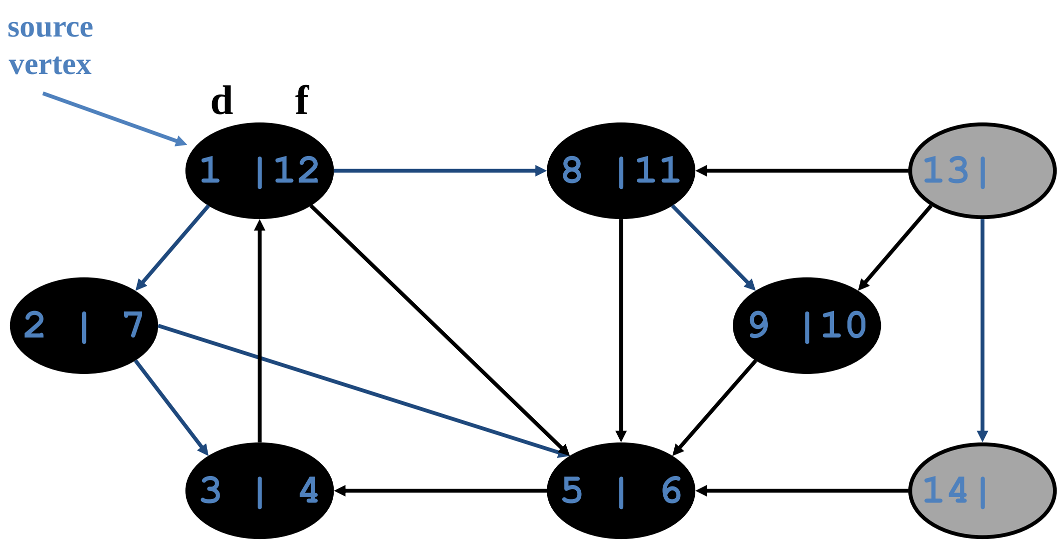
51

DFS Example



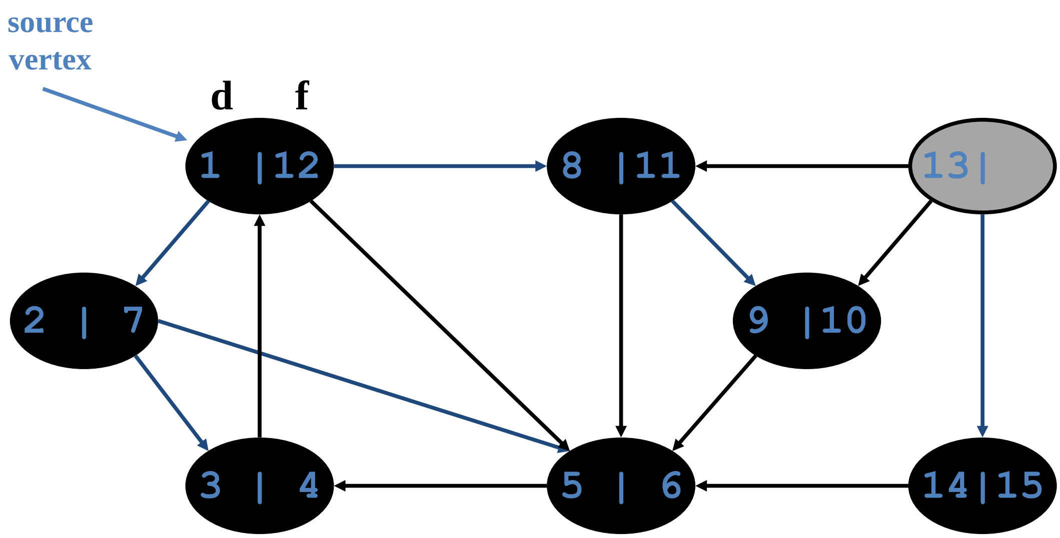
52

DFS Example

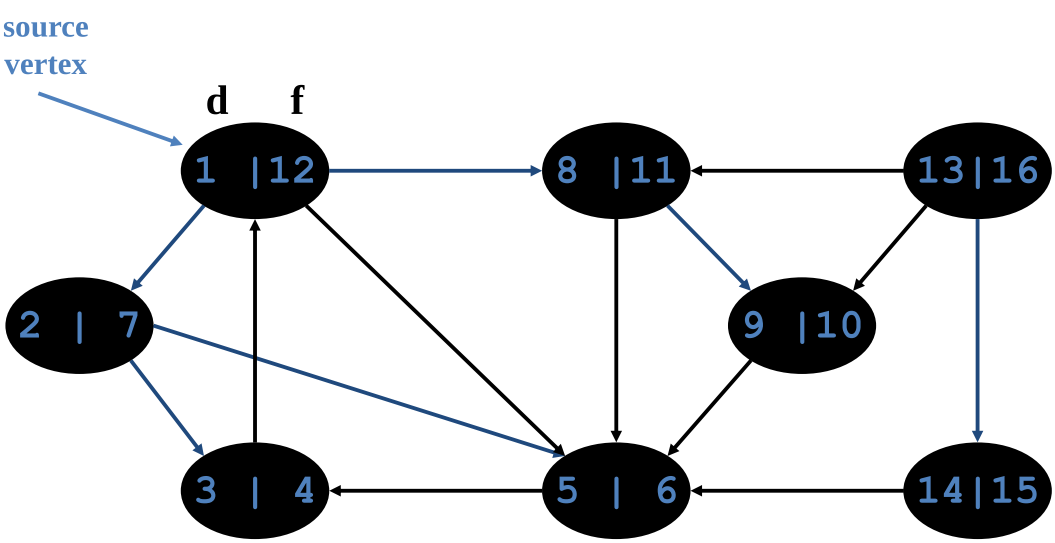


53

DFS Example



DFS Example



55

Reading

Chapter 5 (Sections 5.2, 5.3)
Anany Levitin, Introduction to the design and analysis of algorithms, 3rd Edition, Pearson, 2011.

56